AN ALGORITHM OF HARDWARE UNIT GENERATION FOR PROCESSOR CORE SYNTHESIS WITH PACKED SIMD TYPE INSTRUCTIONS

Yuichiro Miyaoka[†], *Jinku Choi*[†], *Nozomu Togawa*[‡,††], *Masao Yanagisawa*[†], *and Tatsuo Ohtsuki*[†]

[†] Dept. of Electronics, Information and Communication Engineering, Waseda University
3–4–1 Okubo, Shinjuku, Tokyo 169–8555, Japan
E-mail: {miyaoka@ohtsuki, choezg@ohtsuki, yanagi@yanagi, to@ohtsuki}.comm.waseda.ac.jp
[‡] Dept. of Information and Media Sciences, The University of Kitakyushu
1–1 Hibikino, Kitakyushu, Fukuoka 808–0135, Japan
[††] Advanced Research Institute for Science and Engineering, Waseda University
3–4–1 Okubo, Shinjuku, Tokyo 169–8555, Japan
E-mail: togawa@env.kitakyu-u.ac.jp

ABSTRACT

Let us consider to synthesize a processor core with SIMD instructions by a hardware/software cosynthesis system. The system is required to configure functional units executing SIMD instructions and obtain the area and delay of the functional units to evaluate the synthesized processor core. This paper proposes a hardware unit generation algorithm for a hardware/software cosynthesis system of processors with SIMD instructions. Given a set of instructions to be executed by a hardware unit and constraints for area and delay of the hardware unit, the proposed algorithm extracts a set of subfunctions to be required by the hardware unit and generates more than one architecture candidates for the hardware unit. The algorithm also outputs the estimated area and delay of each of the generated hardware units. The execution time of the proposed algorithm is very short and thus it can be easily incorporated into the processor core synthesis system. Experimental results demonstrate effectiveness and efficiency of the algorithm.

1. INTRODUCTION

By modifying a b-bit functional unit, it can execute a single b-bit operation in one case but it can also execute n-parallel b/n-bit sub-operations in another case. For example, a modified 32-bit functional unit can execute four parallel 8-bit operations. A *packed SIMD type operation* (or a *SIMD operation* in short) is n-parallel b/n-bit sub-operations executed by a modified b-bit functional unit. In image processing, each pixel can be represented by an 8-bit data. On the other hand, other operations such as a memory address operation need 32-bit or more bits. Since a modified functional unit with SIMD operations (called a *SIMD functional unit*) can execute both a 32-bit operation and four 8-bit operations, it is effective for image processing. Since an image processor with SIMD functional units executes n pixels concurrently, it can execute application programs of image processing quickly.

An instruction corresponding to a SIMD operation is called a *SIMD instruction.* A SIMD instruction has an operation type, which is a type of a SIMD functional unit executing the instruction. For example, a SIMD instruction executed by a SIMD adding unit has an operation type of *add.* A SIMD instruction can have many options other than an operation type. Let us consider that the result of a b/n-bit operation is over the maximum value represented by b/n bits. In one case, the result can be saturated to the maximum value represented by b/n bits (called a saturation operation) or just high bits of the result can be dicarded (called a wrap around operaion). In another case, the result can be kept as $2 \times b/n$-bit data. It is called a bit-extend operation. Thus for each operation type, a SIMD instruction has many options such as a packing number, whether the data can be signed or unsigned, whether the data can be saturated or wrapped around, whether the data can be bit-extended, and how much the data is shifted. We can have a large number of SIMD instructions. However, if a particular application program runs on an image processor, a small number of SIMD instructions are actually used. We consider that it is important to synthesize an appropriate instruction set of an image processor depending on an application program. Hardware/software codesign can be one of the powerful methodologies in order to synthesize an appropriate instruction set.

We have been developing a hardware/software cosynthesis system for digital signal processor cores[11]. In hardware/software partitioning, a set of instructions in the processor is optimized, area of the synthesized processor core and the execution time of a given application program are estimated, and an appropriate processor core configuration is obtained. Depending on instructions of a processor, a processor core has appropriate *hardware units*, which are functional units, addressing units, and hardware loop units. Area and critical path delay of a hardware unit must be estimated in order to estimate area of a processor core and the execution time of an application program. If SIMD instructions are synthesized depending on an application program, the processor core requires a specific SIMD functional unit to the application program. Since we have many SIMD instructions for each operation type, the number of such a specific SIMD functional unit must be very large. Since we cannot prepare many SIMD functional units in a hardware unit library, we must generate a hardware unit library with estimated area and delay. Besides, area and delay of hardware units should be estimated quickly because area of a processor core and the execution time of the application program for many processor core configurations must be estimated in order to obtain an appropriate processor core configuration.

In our former system[11] or researches on processor synthesis reported such as in [1, 10, 13], hardware units have been designed manually. However, all the SIMD functional units cannot be prepared because they have a large number of configurations. In [5], FHM-DBMS[8], a management system of hardware units is used. In FHM-DBMS, it is difficult to synthesize a functional unit under the constraints of area and delay. Therefore, it is insufficient for hardware/software partitioning.

In this paper, we propose a hardware unit generation system for hardware/software cosynthesis of digital signal processors with SIMD instructions. Given a set of instructions executed by a hardware unit and the constraints of area and delay of the hardware unit, the system generates more than one hardware unit candidates and estimated area and delay of them. An appropriate hardware unit can be selected among hardware unit candidates. The hardware unit generation system is composed of a subfunction extractor and an architecture configurator. A subfunction extractor determines an architecture template based on a set of input instructions. An architecture configurator assigns subfunctional units to subfunctions in the architecture template and outputs hardware unit configurations and estimated area and delay. Especially for the architecture configurator, we propose an architecture configuration algorithm. The algorithm enumerates hardware unit configurations quickly even if any set of instructions is given. The algorithm also can be applied to hardware units other than SIMD functional units.

This paper is organized as follows: Sect. 2 defines a set of SIMD instructions; Sect. 3 proposes a hardware unit generation system; Sect. 4 proposes an architecture configuration algorithm in order to obtain configurations of hardware units quickly; Sect. 5 shows several experimental results including SIMD functional units, addressing units, and hardware units and evaluates effectiveness of the system; Sect. 6 gives concluding remarks.

Table 1: Basic instructions.

Arithmetic and logic operation	ADD, SUB, SRA, SRL, SLL, AND, OR, XOR, MUL, DIV, SLT, SEQ, SNE, COM2, MAC, INC, DEC, ADDI, SUBI, SRAI, SRLI, SLLI, ANDI, ORI, XORI, MULI, DIVI
Load and store	LDX, LDY, STX, STY, LDRX, LDRY, STRX, STRY, LDXI, LDYI, STXI, STYI, LDIX, LDIY, STIX, STIY, MV, IMM
Jump	BEQ, BNE, BZ, BNZ, JP, LOOP, RPT, CALL, RET, NOP, HLT
Parallel load and store	LDPX, STPX

Table 2: SIMD instructions.

Arithmetic operation	ADD, SUB, MUL, MAC
Shift operation	SRA, SLA, SLL
Bit extend/extract operation	EXTD, EXTR
Others	EXCH

2. SIMD INSTRUCTIONS

Target processors for our hardware/software cosynthesis system have a set of instructions in Table 1 and a set of SIMD instructions in Table 2. A SIMD instruction executes n-parallel sub-operations of b/n-bit width with a b-bit functional unit concurrently. Let n be a packing number and k be b/n.

A SIMD instruction has an operation type and several parameters. SIMD instructions have n-parallel operations for k-bit data(Figure 1) and $n/2$-parallel bit-extended operations for k-bit data(Figure 2). Also SIMD instructions can have shift operations and saturation operations for arithmetic operation results. For example, the multiplying instruction with a packing number of 4, signed operation, 2 bits right shift, and saturation is represented by MUL_4_sr2s.

3. HARDWARE UNIT GENERATION SYSTEM

In this section we first discuss relation between hardware/software partitioning and area/delay estimation of hardware units especially in a hardware/software cosynthesis system and then propose a hardware unit generation system.

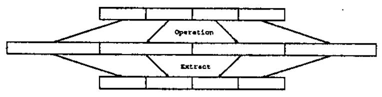

Figure 1: n-parallel operations for k-bit data.

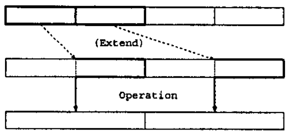

Figure 2: $n/2$-parallel bit-extended operations for k-bit data.

3.1. Hardware/software cosynthesis system and hardware units

Given an application program written in C language and a set of application data, our hardware/software cosynthesis system of digital signal processor cores[11] synthesizes a hardware description of a processor core and generates an object code and a software environment(a compiler and a simulator). The system is mainly composed of compiler, a hardware/software partitioner, a hardware generator, and a software generator.

In hardware/software partitioning, the system obtains an appropriate processor core configuration based on area of a processor core and the execution time of an application program. To explore processor core configurations, the process mainly focuses on hardware units such as functional units, addressing units, and hardware loop units. Thus, area and delay of a hardware unit must be estimated in hardware/software partitioning(Figure 3). Let us consider minimizing area of a processor core. The instruction set of a processor core is reconfigured in order to minimize area of it. If the instruction set of a processor core is reconfigured, a hardware unit in the processor core can be also reconfigured. Area of the reconfigured hardware unit can be different from that of the hardware unit before it is reconfigured. Let us consider that a SIMD functional unit u is reconfigured to a SIMD functional unit u' according as the instruction set of a processor core is reconfigured. Let area of hardware unit u and u' be a_u and $a_{u'}$ respectively. Then, $a_{u'}$ must be less than a_u so that area of the reconfigured processor core can be less than that of the processor core before it is reconfigured. For the execution time of a processor core, the similar discussion holds true. Besides, in order to explore a large number of processor core configurations, a hardware unit should be quickly configured and area and delay of the hardware unit should be estimated.

3.2. Hardware unit generation system

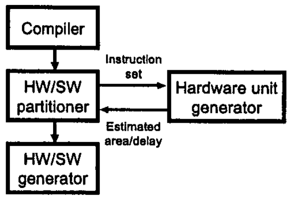

Figure 3: A hardware/software cosynthesis system and hardware unit generation.

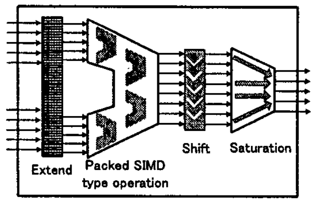

Figure 4: The architecture template of a SIMD functional unit

Based on discussion of section 3.1, we propose a hardware unit generation system.

3.2.1. Architecture template

A hardware unit is realized by several hardware parts which realizes a part of hardware unit functions (called a *subfunctional unit*). A *subfunction* is a function realized by a subfunctional unit. An *architecture template* represents a set of subfunctions and their connections. Area of a hardware unit is estimated by adding up area of subfunctional units assigned to subfunctions in an architecture template and area of a controller depending on the architecture template. Delay of hardware unit is estimated by the critical path delay of subfunctional units on the critical path of a hardware unit depending on an architecture template and delay of a controller.

For example, a hardware unit which executes `MUL_4_sr3s` and `MUL_4h_sl5` has an architecture template with subfunctions and their connections in Figure 4. The subfunctions are bit extension of 2 higher packing data, signed multiplication and extended multiplication of 4 packing data, 3 bit right shift and 5 bit left shift, and signed saturation. Figure 5 shows architecture templates of an addressing unit and a hardware loop unit. Figure 6 shows that several hardware unit configurations are obtained by assigning subfunctional units to subfunctions in an architecture template.

3.2.2. Overview of our system

Given a set of instructions executed by a hardware

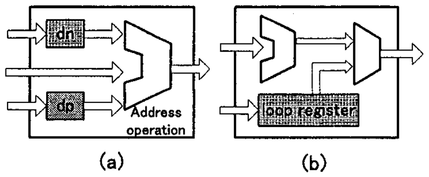

Figure 5: (a) The architecture template of an addressing unit. (b) The architecture template of a hardware loop unit.

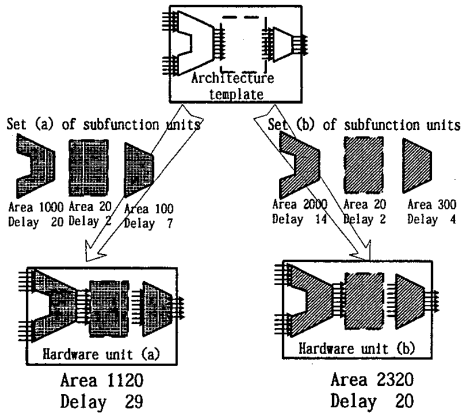

Figure 6: Assignment of subfunction units to subfunctions in an architecture template.

unit and constraints of area and delay of the hardware unit, a hardware unit generation system generates more than one hardware unit candidates and outputs their estimated area and delay. Figure 7 shows the proposed system. The hardware unit generation system is composed of a subfunction extractor and an architecture configurator. A subfunction extractor determines an architecture template based on a set of input instructions. An architecture configurator assigns subfunctional units to subfunctions in the architecture template and outputs hardware unit configurations and their estimated area and delay.

Since a hardware unit is realized by a combination of subfunctional units, the system can generate hardware units for any given set of instructions by preparing a small number of subfunctional units. Since the system can give constraints of area and delay to hardware units, the system can outputs only the hardware units required in hardware/software partitioning which intends to reduce area of a processor core or the execution time of an application program. Since the system can enumerate more than one candidates, a hardware unit for a set of instructions need not be configured

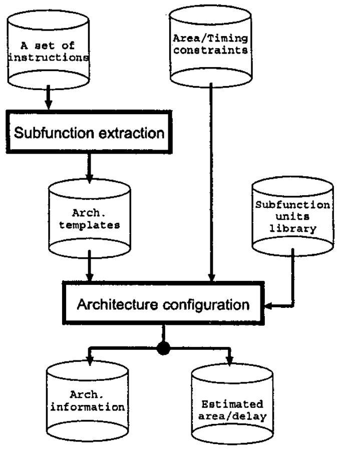

Figure 7: The hardware unit generation system.

many times. Thus, hardware/software partitioning can be quickly performed.

4. ARCHITECTURE CONFIGURATION ALGORITHM

This section defines an architecture configuration problem which is the core of the hardware unit generation system and proposes an architecture configuration algorithm which enumerates hardware unit configurations.

4.1. Architecture configuration problem

Let I_u be a set of instructions executed by a hardware unit u. Let a_u be area of a hardware unit u and a_{max} be the input maximum area of a hardware unit. Then an *area constraint* is given by $a_u \leq a_{max}$. Let d_u be critical path delay of a hardware unit u and d_{max} be the input maximum delay of a hardware unit. Then a *delay constraint* is given by $d_u \leq d_{max}$. Given an architecture template which depends on a set I of instructions, an area constraint a_{max} and a delay constraint d_{max}, an architecture configuration problem is to obtain more than one hardware unit configurations which satisfy $I \subseteq I_u$, an area constraint, and a delay constraint.

4.2. Algorithm

Let $F = \{f_1, f_2, \ldots, f_m\}$ be a set of subfunctions in an architecture template. We can select a set S_i of subfunctional units which can realize f_i $(i = 1, \ldots, m)$. A subfunctional unit $s \in S_i$ has area $a(s)$ and delay $d(s)$.

In an architecture configuration problem, more than one candidates must be enumerated quickly. It is desirable that enumerated hardware unit configurations have smaller area in hardware units which have the same delay. Therefore we propose the algorithm as follows.

First, we assign a subfunctional unit which has the shortest delay in S_i ($i = 1, \ldots, m$) to each subfunction f_i. The generated hardware unit has the shortest delay in all the candidates and are expects to satisfy a delay constraint.

Second, we focus on a subfunction f_i and try to replace the subfunctional unit s currently assigned f_i with a subfunctional unit $s' \in S_i \backslash s$ such that $a(s') < a(s)$ and $d(s')$ is minimum. We perform this operation for each of all the subfunctions and actually replace s with s_{min} which satisfies a delay constraint and has the smallest area of all the generated configurations. This algorithm can obtain hardware unit configurations which have small area gradually by repeating this process while a hardware unit satisfies a delay constraint.

Lastly, we enumerate hardware unit configurations which satisfy an area constraint.

Figure 8 shows this algorithm. Let us calculate time complexity of the proposed algorithm. Let n be the total of subfunctional units $\sum|S_i|$. If subfunctional units which realize a subfunction f_i are sorted on delay in ascending order, Step 2 of Figure 8 can be executed with $O(1)$. Since Step 3 is executed for all subfunctions, Step 2–4 can be executed with $O(m)$. Step 2–4 are repeated at most n times. Thus, time complexity is $O(mn + n \log n)$. Let us calculate space complexity of the proposed algorithm. This algorithm need to keep n sorted subfunctional units. A hardware unit is realized by m subfunctional units. Thus, space complexity is $O(m + n)$.

Step 1 Configure a hardware unit assigning each subfunctional unit which has the shortest delay to each subfunction of an architecture template. If the configured hardware unit does not satisfy a delay constraint, finish.

Step 2 Output the current configuration if the current configuration of a hardware unit satisfy an area constraint.

Step 3 Let S_i be a set of subfunctional units assigned to a subfunction f_i. Let a(s) and d(s) be area and delay of a subfunctional unit $s \in S_i$. For a subfunction f_i, try to replace the subfunctional unit s currently assigned f_i with a subfunctional unit $s' \in S_i \backslash s$ such that $a(s') < a(s)$ and $d(s')$ is minimum.

Step 4 For each of all the subfunctions, perform Step 3. Actually replace s with s_{min} which satisfies a delay constraint and has the smallest area of all the generated configurations. Update S_i to $S_i \backslash s_{min}$.

Step 5 Repeat Step 2–4 while there exists a hardware unit which satisfies the condition written in Step 4.

Figure 8: The proposed algorithm.

5. EXPERIMENTAL RESULTS

The system has been implemented in the C language on Sun Ultra SPARC(200MHz). The system was applied to a SIMD multiplier and an addressing unit. We used gcc(version 2.95.2) to compile. We described subfunctional units in VHDL and logic-synthesized them using Synopsys Design Compiler with the VDEC cell libraries(CMOS and 0.35μm technology). [1] Thus we obtained area and delay of the subfunctional units. We enumerated all the hardware unit configurations.

Figure 9 shows the results of a SIMD multiplier given a set of instructions in Table 3 and constraints of area and delay as 900,000[μm^2] and 16.0[ns] respectively. The number of hardware unit configurations enumerated by our proposed algorithm is about 20% of that enumerated by a full search algorithm. However, our algorithm can enumerate most of configurations which have the smallest area of all the hardware units with the same delay. Our proposed algorithm and a full search algorithm can obtain the results of Figure 9 by 0.17 [ms] and 2.10[ms], respectively. In a hardware/software partitioning, a hardware unit generation system must be executed hundred thousands of times so that more processor core configurations can be explored. Since our algorithm can obtain the configurations more than 10 times faster than a full search algorithm, hardware/software partitioning can be performed quickly and more processor core configurations can be explored.

Table 3: The input set of instructions for a SIMD multiplier.

MUL_2_sw	MUL_2_ss
MUL_4_sw	MUL_4_ss
MUL_2_sr7w	MUL_2_sr4s
MUL_4_sr4w	MUL_4_sr4s

Next, we can apply our algorithm to an addressing unit[6, 7]. Table 4 shows functions of a generated addressing unit. Figure 10 shows the results given constraints of area and delay as 280,000[μm^2] and 20[ns] respectively. This results show that our proposed algorithm can effectively obtain configurations for general hardware units as well as SIMD functional units.

Therefore, we can obtain a better configuration of processor cores in hardware/software partitioning by using our hardware unit generation system.

[1]The libraries in this study have been developed in the chip fabrication program of VLSI Design and Education Center(VDEC), the University of Tokyo with the collaboration by Hitachi Ltd. and Dai Nippon Printing Corporation.

Table 4: The input set for an addressing unit.

The functions	
`post increment` `index add`	`post decrement` `modulo add`
The number of address registers	8
The number of index register	8

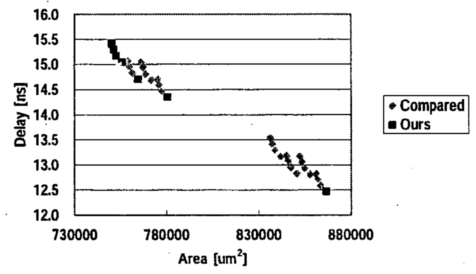

Figure 9: The results of the SIMD multiplier.

6. CONCLUSION

This paper proposed a hardware unit generation system for hardware/software cosynthesis system of processor cores with SIMD instructions. The experimental results demonstrate that the system can enumerate more than one configurations which have small area and short delay. In the future, we will incorporate constraints of power dissipation or the number of cycles for multicycle functional units into our system.

Acknowledgement

The authors would like to thank Nobuhiro Nonogaki (presently with Toshiba Co., Ltd.) for their valuable discussions.

REFERENCES

[1] H. Akaboshi and H. Yasuura, "COACH: A computer aided design tool for computer architectures," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E76-A, No. 10, pp. 1760–1769, 1993.

[2] C. Basoglu, W. Lee, and J. S. O'Donnell, "The MAP 1000A VLIW mediaprocessor," *IEEE Micro*, Vol. 20, No. 2, pp. 48–59, 2000.

[3] K. Diefendorff, P. K. Dubey, R. Hochsprung, and H. Scales, "AltiVec extension to PowerPC accelerates media processing," *IEEE Micro*, vol. 20, no. 2, pp. 85–94, 2000.

[4] S. Ishiwata and T. Sakurai, "Future directions of media processors," *IEICE Trans. on Electronics*, Vol. E81-C, No. 5, pp. 629–635, 1998.

[5] M. Itoh, S. Higaki, J. Sato, A. Shiomi, Y. Takeuchi, A. Kitajima, and M. Imai, "PEAS-III: An ASIP design environment," in *Proceedings of the 2000 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, pp. 430–436, 2000.

[6] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee, *Processor Fundamentals: Architectures and Features*, Berkeley Design Technology, Inc., 1994–1996.

[7] V. K. Madisetti, *Digital Signal Processors*, IEEE Press, 1995.

[8] T. Morifuji, Y. Takeuchi, J. Sato, and M. Imai, "Flexible hardware model database management system: implementation and effectiveness," in *Proc. of the Synthesis and System Integration Mixed Technologies(SASIMI'97)*, pp. 83–89, 1997.

[9] A. Peleg and U. Weiser, "MMX technology extension to the Intel architecture," *IEEE Micro*, Vol. 16, No. 4, pp. 42–50, 1996.

[10] J. Sato, A. Y. Alomary, Y. Honma, T. Nakata, A. Shiomi, N. Hikichi, and M. Imai, "PEAS-I: A hardware/software codesign system for ASIP development," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E77-A, No. 3, pp. 483–491, 1994.

[11] N. Togawa, M. Yanagisawa, and T. Ohtsuki, "A hardware/software cosynthesis system for digital signal processor cores," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E82-A, No. 11, 1999.

[12] M. Tremblay, J. M. O'Connor, V. Narayanan, and L. He, "VIS speeds new media processing," *IEEE Micro*, Vol. 16, No. 4, pp. 10–20, 1996.

[13] J.-H. Yang, B.-W. Kim, S.-J. Nam, Y.-S. Kwon, D.-H. Lee, J.-Y. Lee, C.-S. Hwang, Y.-H. Lee, S.-H. Hwang, I.-C. Park, and C.-M. Kyung, "MetaCore: An application-specific programmable DSP development system," *IEEE Trans. on Very Large Scale Integration(VLSI) systems*, vol. 8, no. 2, pp. 173–183, 2000.

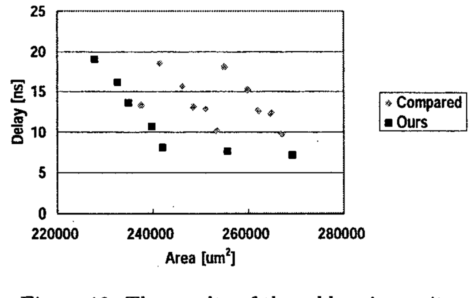

Figure 10: The results of the addressing units.